Load-Balancing and Prudent Deployment of VNFs for Heterogeneous Multicore Systems

Jung-Chun Kao, Guang-Han Ma, Cheng-Yu Lee, Chun-Fu Kuo, and Jia-Hong Hong
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan

***Abstract*—Network function virtualization (NFV) is an enabling technology for telecommunication operators to achieve flexible service deployment and efficient reuse of physical resources; operators are no longer restricted to dedicated hardware and can run their services on off-the-shelf servers. However, there has been little research effort on optimizing NFV for heterogeneous multicore architectures such as ARM's big.LITTLE technology and Intel's Performance Hybrid Architecture. In this paper, we consider CPU core affinity of virtual network functions (VNFs) and study load-balancing and prudent deployment of VNFs in a heterogeneous multicore system. We not only develop the Four-Class Filling (FCF) algorithm with a complexity of $O(n \log n)$ and an approximation ratio of 3/4, but also implement it on an NFV platform equipped with two types of CPU cores. Real-world measurement validates feasibility and effectiveness of our method. Besides, simulation results show that our algorithm performs very well in terms of load-balancing factor, operational cost, and excess usage.**

Index Terms—Network function virtualization, VNF deployment, heterogeneous multicore system, software-defined network

I. Introduction

Network function virtualization (NFV) represents a paradigm shift in telecommunication. Traditionally, individual network functions such as routers and firewalls execute on dedicated hardware. The emergence of NFV enables to move network functions out of dedicated hardware and into software modules that run on commodity machines like off-the-shelf servers. Such software modules are called virtual network functions (VNFs) and often employ virtualization technologies, running within containers or virtual machines.

Since NFV decouples network functions from underlying dedicated hardware devices, network service providers or operators can easily add, change, or migrate VNFs on commodity machines. The ability to provide network services on demand and in real time at low infrastructure cost facilitates operators to achieve flexible service deployment and efficient reuse of physical resources.

A service function chain (SFC) is a sequence of multiple VNFs in which specific flows are steered and processed in order. Allocating the VNFs in SFCs to physical resources plays an important role in NFV, because it affects performance significantly. There has been a considerable amount of related work in the literature. For example, [10] studies the VNF Orchestration Problem (VNF-OP) aiming to minimize the overall network operational cost by placing the VNFs in a given set of SFCs to physical nodes such that delay and capacity constraints are met. VNF-OP, which is formulated in [10] as an Integer Linear Programming (ILP) problem, is NP-hard.

Using the sharing capability enabled by NFV, [9] addresses the VNF forwarding graph (VNF-FG) placement and chaining problem. In [9], VNF-FG is formulated as an ILP problem whose main objective is to minimize the total consumed power; scalability concern due to NP-harness is overcome by limiting the number of candidate hosts. In [11], a method based on Monte Carlo tree search is developed for energy consumption minimization. [14] leverages graph convolutional network and deep reinforcement learning to minimize a weighted sum of energy consumption and end-to-end delay of SFCs.

In addition to optimization of operational expenditure and power/energy consumption, improving utilization of physical resources has also attracted plenty of attention. Given a set of SFCs and their injection rates, [12] aims to minimize the average usage (in percentage) of network resources including communication links and compute nodes. [13] develops an VNF deployment algorithm for scaling-out, which tends to deploy the VNFs of new SFC requests to the physical machines that have already been deployed VNFs in order to utilize as few physical machines as possible.

None of the above work is designed for heterogeneous multicore architectures. Heterogeneous multicore architectures bundle cores of different types into a single physical processor; an example is Intel's Performance Hybrid Architecture, which integrates two types of cores—faster yet more power-hungry Performance-cores (P-cores) and slower yet more power-efficient Efficient-cores (E-cores). Another example is ARM's big.LITTLE technology. Processors with heterogeneous cores have been widely used in many devices such as smart phones and computers; the number of cores per processor have been increasing steadily, from single digit to double digits or even hundreds. NFV can benefit from such hybrid and strong computing power. However, there has been little effort taken in this research direction, which motivates us to study NFV with heterogeneous multicores considered explicitly.

In this paper, we focus on load-balancing and prudent deployment of affinity-aware VNFs for heterogeneous multicore systems. Although our method can be extended to a system with three or more types of cores, we focus on a system with cores of two types, say P-cores and E-cores, for two-fold reasons: First, the majority of heterogeneous multicore CPUs available in the market are equipped with two types of cores; second, it is for simplicity of exposition.

979-8-3503-0358-2/24/$31.00 ©2024 IEEE

As we measure and show in Fig. 1, the end-to-end latency of a compute-bound SFC running on a P-core is quite different from the SFC latency running on an E-core. This implies the importance of affinity awareness: A VNF that either is computationally intensive or has a strict requirement for latency must run on a P-core. In addition to affinity awareness, we also consider load-balancing and prudence, which aims to distribute the workload of a given set of VNFs to multiple cores for better responsiveness, while preventing cores from being underused. VNFs deployed to underused cores should be re-deployed to fewer cores to reduce the operational cost. To achieve the above requirements in real time, we propose an algorithm called Four-Class Filling (FCF) with a time complexity of $O(n \log n)$ and an approximation ratio of 3/4 under some condition.

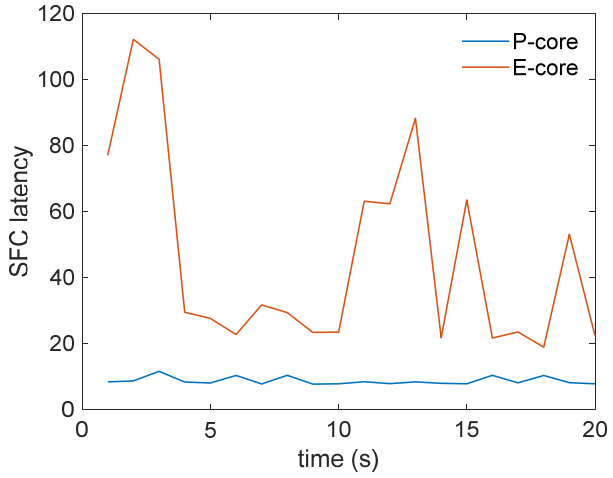

Fig. 1. The end-to-end latency of a compute-bound SFC we measure.

The remainder of this paper is organized as follows. In Section II, we formulate the load-balancing and prudent VNF deployment problem for a heterogeneous multicore system. We present the Four-Class Filling (FCF) algorithm we develop for the VNF deployment problem in Section III. Section IV presents both measurement results of our implementation and simulation results of our algorithm. We present concluding remarks in Section V.

II. Problem Formulation

The problem we consider in this paper is load-balancing and prudent VNF deployment tailored to a heterogeneous multicore system. Roughly speaking, we take the advantage of affinity awareness on such a system and aim to distribute the workload of VNFs evenly to multiple cores (for better responsiveness), while not utilizing too many cores (for lower operational cost). For simplicity of exposition, we focus on a heterogeneous multicore system with cores of two types, although our method can be extended to a system with more than two types of cores.

We consider a heterogeneous multicore machine that has a set of available E-cores (denoted by $E = \{1,2,\dots,|E|\}$) and a set of available P-cores (denoted by $P = \{|E|+1, |E|+2, \dots, |E|+|P|\}$). In total, there are $|O| = |E| + |P|$ available cores, which is denoted by $O = \{1,2,\dots,|O|\}$. A P-core has higher computing power than an E-core. The (computing) *capacity* of core o is

$$c_o = \begin{cases} C_{\max} & \text{if core } o \text{ is a E-core} \\ k \cdot C_{\max} & \text{if core } o \text{ is a P-core} \end{cases} \quad (1)$$

where k is a constant greater than one.

We are given a set of VNFs, each of which is required to be deployed to one of the available cores under a few constraints. The set of these VNFs is denoted $N = \{1,2,\dots,|N|\}$. Each VNF has its own size or called *workload*, which is the amount of computing power it needs. The workload of VNF n is denoted by $s_n, n \in N$.[1] A larger VNF means its workload is heavier.

Each VNF must be deployed to a core with enough residual capacity. In addition, if a VNF is computationally intensive or it has a strict requirement for latency and cannot fulfill by a E-core, the VNF must be deployed to a P-core. Such a VNF is called *P-affined*. The usage of each core cannot exceed the core's capacity. The *usage* of core o is denoted by u_o, $o \in O$. For any $n \in N$, deploying VNF n to a core increases the usage of the core by s_n and decreases the residual usage by s_n.

According to their usage, cores are categorized into unused cores and used cores; used cores are further categorized into underused cores and well-used cores. A used core is a core with positive usage and an unused core is a core with zero usage. An underused core is a used core whose usage is below a threshold, whereas a well-used core is a used core whose usage is equal or greater than the threshold. The usage *threshold* of core o, $o \in O$, is set to be:

$$t_o = \begin{cases} C_{\min} & \text{if core } o \text{ is a E-core} \\ k \cdot C_{\min} & \text{if core } o \text{ is a P-core} \end{cases} \quad (2)$$

The *margin* of a core is defined as the threshold value of the core subtracted by its usage. Positive margin implies underused or unused, whereas non-positive margin implies well-used.

VNF deployment should not result in multiple underused cores. Having many underused cores implies plenty of residual capacity and hence it is easy to re-deploy VNFs to fewer cores. Using a reduced number of cores, CPUs, or virtual machine instances can reduce the operational cost. Once VNF deployment is done, there should exist zero or one underused core. This requirement is referred to as the *prudence constraint*.

For load-balancing, all VNFs in N should be deployed to as many cores as possible so as to distribute the workload of these VNFs evenly. Combining load-balancing and prudence, all VNFs in N should be deployed to as many cores as possible while keeping all (except at most one) of these cores well-used.

The *VNF deployment problem* we study in this paper aims to maximize the number of cores to which a given set of VNFs are deployed, while satisfying *1)* the usage of each core cannot exceed its capacity, *2)* the number of underused cores is at most one, and *3)* all the P-affined VNFs must be deployed to P-cores. Equivalently, we aim to maximize the number of well-used cores, while having at most one underused core and deploying

[1] We assume that s_n is smaller than the threshold of a P-core, for each $n \in \mathbb{N}$. Otherwise, an exclusive P-core is allocated to the VNF for load-balancing.

all P-affined VNFs to P-cores. The VNF deployment problem we study is a modified version of the bin covering problem. The differences of these two problems are three-fold: First, the VNF deployment problem involves two types of cores/bins (i.e., E-cores and P-cores), whereas the bin covering problem has a single type of bins. Second, the usage of each core in the VNF deployment problem is bounded below (by the threshold) and bounded above (by the capacity), whereas the bin size in the bin covering problem is only bounded from below. Third, the bin covering problem does not consider affinity.

III. Our Algorithm: Four-Class Filling

To solve the load-balancing and prudent VNF deployment problem in an affinity-aware, fined-grained, and efficient way, we propose the four-class filling (FCF) algorithm, which consists of three stages—preprocessing, filling, and postprocessing. What follows present how FCF works, as well as the complexity and the approximation ratio of FCF.

A. The Preprocessing Stage

Initially, the usage of each core is zero; that is, $u_o = 0, o \in O$. All VNFs in N are sorted and numerated in the descending order of their workload. Based on the workload, all VNFs are divided into the following four classes:

$$\begin{aligned} W &= \{n \in N: C_{\min} \le s_n \le k \cdot C_{\min} \text{ or } n \text{ is P-affined}\} \\ X &= \left\{n \in N: \tfrac{1}{2} C_{\min} \le s_n < C_{\min}\right\} \\ Y &= \left\{n \in N: \tfrac{1}{3} C_{\min} \le s_n < \tfrac{1}{2} C_{\min}\right\} \\ Z &= \left\{n \in N: 0 < s_n < \tfrac{1}{3} C_{\min}\right\} \end{aligned} \tag{3}$$

Note that a P-affined VNF is categorized into *W*-class, regardless of its workload. The VNFs in these four classes are sorted in the descending order of their workload, respectively.

The preprocessing stage pre-allocates *W*-class VNFs to P-cores. An unused P-core is allocated the VNFs in W in order, one at a time, until either the core's margin is no greater than $C_{\min}$ or the core's residual capacity is smaller than the next VNF's workload. If there are *W*-class VNFs unallocated, the same allocation process repeats to a new unused P-core.

The preprocessing stage ends up leaving four types of cores—underused P-cores, (unused) E-cores, well-used P-cores, and unused P-cores. The sets of cores of these four types are denoted by P_L, E, P_H, and P_0, respectively. The filling stage will deal with cores of the first two types. There is no need to fill well-used P-cores with extra VNFs[2]. Utilizing unused P-cores is deferred to the postprocessing stage[3], if necessary.

B. The Filling Stage

The filling stage allocates the VNFs in $N' = X \cup Y \cup Z$ to the cores in $O' = P_L \cup E$, iteratively. An iteration deals with the core in O' with highest priority. During an iteration, each VNF allocated to the core is removed from X, Y or Z; at the end of an iteration, the highest-priority core is removed from O'. The priority of cores is set as follows: Underused P-cores have higher priority than unused E-cores[4]; among the cores of the same type, larger residual capacity implies higher priority. An iteration has two phases. If $X \cup Y \neq \emptyset$ and $Z \neq \emptyset$, an iteration starts with phase 1; otherwise, phase 1 is skipped and only phase 2 is executed.

Phase 1 first does *XY*-filling repeatedly: The highest-priority core is filled with either the largest VNF in X or the two largest VNFs in Y, whichever keeps margin positive and whichever is larger. This *XY*-filling process repeats until either the next *XY*-filling will make the core's margin non-positive or $X \cup Y = \emptyset$. After that, phase 1 does reverse *Z*-filling repeatedly: The core is filled with the VNFs in Z, one at a time, in the ascending order of workload, until *1)* the core's margin becomes non-positive, *2)* the core's residual capacity is too small to accommodate the next VNF, or *3)* $Z = \emptyset$. In the first two cases, this iteration completes with no need to execute phase 2; in the last case, this iteration moves forward to phase 2.

Phase 2 executes only if either $X \cup Y$ or Z is empty. According to the sizes of $X \cup Y$ and Z, phase 2 has three conditional branches:

- $X \cup Y = \emptyset$ and $Z \neq \emptyset$: *Z*-filling is applied to the highest-priority core repeatedly. The core is filled with the VNFs in Z in the descending order of workload, one at a time, until either *1)* the core's margin becomes non-positive, *2)* the core's residual capacity is too small to accommodate the next VNF, or *3)* $Z = \emptyset$.
- $Z = \emptyset$ and $X \cup Y \neq \emptyset$: *XY*-filling is applied to the highest-priority core repeatedly. The core is filled with either the two largest VNFs in X or the three largest VNFs in Y, whichever makes the usage no larger than the capacity and whichever is smaller. *XY*-filling repeats until either the core's margin becomes non-positive or $X \cup Y = \emptyset$. In the case when *XY*-filling cannot allocate any VNF to the core, we switch to use the SI algorithm [1] to fill the core with extra VNFs in $X \cup Y$.
- $X \cup Y = Z = \emptyset$: There is no need to do anything, because all VNFs in $X \cup Y \cup Z$ have been allocated to cores.

After running anyone of the branches, phase 2 completes and so does this iteration.

The filling stage ends up either having all VNFs allocated or having a non-empty set of unallocated VNFs, which is denoted by N''. In the latter case, all cores in $O' = P_L \cup E$ have become well-used during the filling stage; only the unused P-cores (i.e., the cores in P_0) should be used to allocate the VNFs in N''.

C. The Postprocessing Stage

If $N'' \neq \emptyset$ (because there are VNFs in $X \cup Y \cup Z$ that remains unallocated at the end of the filling stage), the postprocessing stage allocates the VNF in N'' to the cores in $O'' = P_0$ in the same way as the filling stage, except the VNFs in N'' are re-categorized into the following three classes

$$\begin{aligned} X'' &= \left\{n \in N'': \tfrac{1}{2} k \cdot C_{\min} \le s_n < k \cdot C_{\min}\right\} \\ Y'' &= \left\{n \in N'': \tfrac{1}{3} k \cdot C_{\min} \le s_n < \tfrac{1}{2} k \cdot C_{\min}\right\} \end{aligned} \tag{4}$$

[2] This is because well-used cores have satisfied all constraints.

[3] This is because we tend to keep as many P-cores unused as possible. They are reserved for future use, especially for P-affined VNFs.

[4] This is because it is not allowed to have multiple underused cores in the VNF deployment problem.

$$Z'' = \left\{ n \in N'' : 0 < s_n < \frac{1}{3} k \cdot C_{\min} \right\}$$

instead of *X*-class, *Y*-class, and *Z*-class shown in (3).

D. Complexity and Approximation Ratio

Due to the space problem, we present the computational complexity and the approximation ratio of FCF without delving into too many details.

The complexity of the pre-processing stage is $O(|N| \cdot \log|N|)$ because of sorting. The complexity of the filling stage and the post-processing stage is $O(|N'|)$ and $O(|N''|)$, respectively, because both XY-filling and Z-filling allocate at least one VNF to a core at a time. Knowing the complexity of FCF's stages, it is straightforward to get the complexity of the entire FCF, as shown in Theorem 1.

Theorem 1. FCF is of complexity $O(n \log n)$, where $n = |N|$ is the total number of VNFs.

Due to the space problem, we present the approximation ratio of FCF without showing any derivation.

Theorem 2. If $W = \emptyset$ and $C_{\max} \geq 2 \cdot C_{\min}$, FCF has an approximation ratio of 3/4.

IV. Performance Evaluation

As will be shown in Section IV.A, we validate the effectiveness of our implementation and our proposed FCF algorithm through a small-scale, real-world experimental measurement. In addition, as will be shown in Section IV.B, by simulation we compare the performance of our proposed FCF algorithm with an existing algorithm for scenarios at medium scale.

A. Experimental Implementation and Measurement

1) Implementation Overview

As shown in Fig. 2, our experiment has three computers, each equipped with one or two 10 gigabit network interface cards (NICs). One of the three computers is a server, which provides network services; the other two computers act as traffic generators. The server equipped with Intel Core i7-12700 processor (which has eight P-cores and four E-cores) executes our proposed FCF algorithm and OpenNetVM [3] to manage deployment and execution of VNFs. The traffic generators inject packet flows to the server, by using Pktgen [4].

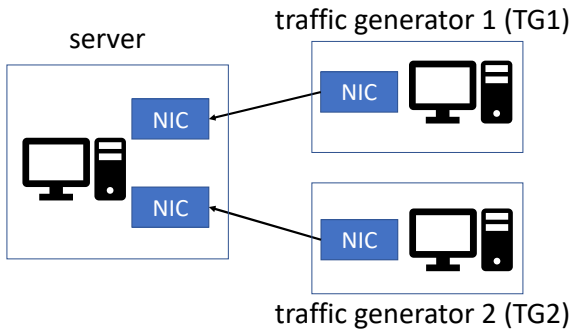

Fig. 2. Overview of our hardware implementation.

On the server, we utilize a technology called process isolation to limit, account for, and isolate the running VNFs. The injection rate and the end-to-end latency of each SFC are measured by a software module we develop in [5]. Based on the injection rate measured, the workload of each VNF is predicted, in real time. VNF workload is predicted by a supervised-learning-based method we devise in [2]. To obtain a training dataset required by supervised learning, we develop an instruction-level tool to collect instruction statistics, Intel Performance Counter Monitor (PCM) [15] metrics, and usage of cores (with the help of a runtime code manipulation system called DynamoRIO [16]).

2) Experimental Measurement

In the experiment, all packets injected to the server are processed through an SFC consisting of two VNFs; packets first go to the *Busy forward* VNF and then move to the *AES encryption* VNF. *Busy forward* is a VNF we develop (by modifying the VNF available in [6]) for workload customization; *AES encryption* is an open-sourced VNF available in [7]. By process isolation technology, two P-cores and two E-cores are reserved for the two VNFs; other processes are not allowed to use these cores.

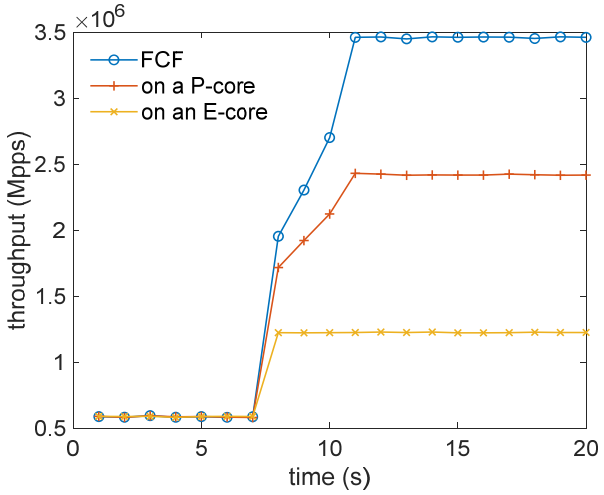

Fig. 3. Throughput measured. The packet injection rate keeps low by the 7th second and then rises gradually to a high rate.

We compare the FCF algorithm with two fixed schemes—VNFs on a P-core and VNFs on an E-core. As shown in Fig. 3, it is observed that at a low injection rate (by the 7th second), these three schemes perform the same because even a single E-core can afford the low workload of the two VNFs. As packets are injected at a high rate, the workload exceeds the capacity of a single core and therefore these three schemes perform differently: FCF performs best because it allocates the two VNFs to two cores and hence the workload is distributed. On the contrary, putting all VNFs on an E-core performs worst because there is no load-balancing and an E-core is slower than a P-core.

B. Simulation Results

By simulation, we compare the performance of our proposed FCF algorithm with the local search algorithm (LSA) [8]. The

LSA first assigns VNFs to cores randomly and then iteratively move a VNF from the busiest core to the lowest-usage core if the usage of the new busiest core (after this movement) is smaller than that of the busiest core in the previous iteration.

The performance metrics of interest include the load-balancing factor, the number of used cores, and the sum of excess usage. The load-balancing factor is defined as the number of well-used cores subtracted by the number of underused cores, in order to reflect that well-used cores are beneficial from the load-balancing aspect and underused cores are costly from the prudence aspect. The excess usage of core o is defined as $(u_o - t_o)^+$, which is the amount of usage that exceeds the threshold value. The sum of excess usage is the total amount of excess usage of all cores, which is $\sum_{o \in O} (u_o - t_o)^+$. Unlike Section IV.A, throughput is not within the scope of interest in this subsection; this is because the total workload of VNFs is set to be less than the total capacity of cores and thus all the algorithms used in this paper result into the same throughput in the long run.

In the simulation, the number of E-cores is fixed at 8, but the number of the P-cores varies: The scenarios with 4, 6, and 8 P-cores are considered.[5] The number of instances per scenario is 1000. $C_{\max}$ is set/normalized to 100, $C_{\min}$ is set to 50, and k is set to 1.5. In each scenario, the number of VNFs per service function chain (SFC) is set to 5 and the number of SFCs is set to be equal to the number of P-cores. Among the VNFs, the number of *W*-class VNFs is half of the number of P-cores, the number of *X*-class VNFs is equal to the number of P-cores, the number of *Y*-class VNFs is equal to the number of P-cores, and the remaining VNFs are all *Z*-class VNFs. The workload of VNFs in each class is uniformly distributed in-between the lower bound and the upper bound defined in (3).

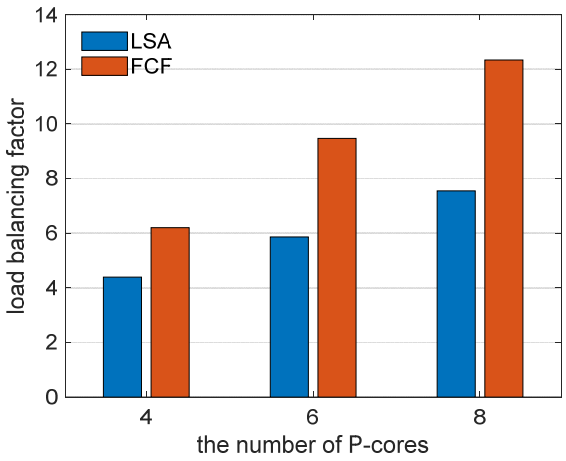

Fig. 4. Load-balancing factor.

Fig. 4 shows the simulation results in terms of the (averaged) load-balancing factor; the larger load-balancing factor, the better. It is observed that our proposed FCF algorithm consistently outperforms LCA in load-balancing factor, regardless of the number of P-cores.

In terms of the number of used cores, Fig. 5 shows that LSA consumes all P-cores and all E-cores in all the three scenarios. However, a considerable number of cores consumed under LSA are underused; this explains why LSA performs poorly in terms of load-balancing factor. On the contrary, FCF allocates VNFs to less cores that LSA does. A reason behind is that FCF considers both load-balancing and prudence, which increases the number of well-used cores but prevents cores from becoming underused; on the contrary, LSA tends to use all the cores. From the point of view of operational cost, FCF outperforms LSA.

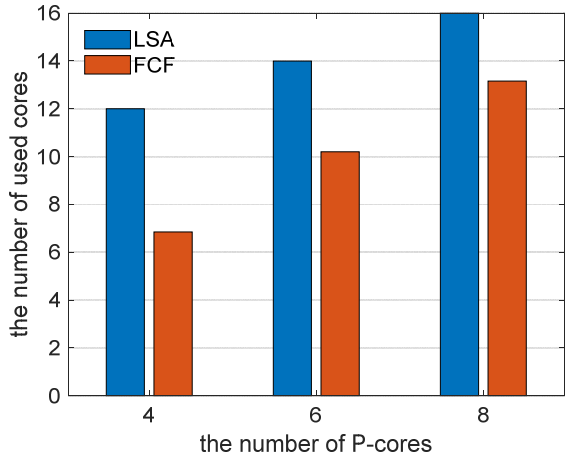

Fig. 5. The number of used cores.

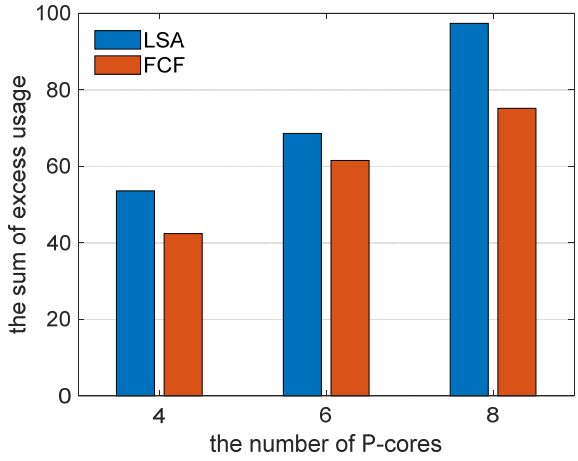

Fig. 6. The sum of excess usage.

We also show the simulation results in terms of the sum of excess usage; the smaller excess usage, the better. In contrast to load-balancing factor which is a discrete indicator (because a used core is either well-used or underused), the sum of excess usage is a continuous indicator (that reflects the total amount of

[5] These numbers of cores are in accordance with Intel Core i9-12900 processor, which has eight E-cores and eight P-cores.

excess usage compared to the ideal case in which the usage of each used core is equal to its threshold value). As shown in Fig. 6, FCF performs better than LSA, because the sum of excess usage caused by FCF is smaller than that of LSA. A summary can be drawn from what is observed in Fig. 4, Fig. 5, and Fig. 6. Compared with LSA, FCF not only distributes the workload over used cores more evenly, which implies better load-balancing, but also uses fewer cores, which results in better prudence (and lower operational cost).

V. Conclusion

This paper studies the load-balancing and prudent VNF deployment problem for heterogeneous multicore systems. We consider a commodity machine equipped with two types of cores and develop a fast yet efficient algorithm. The FCF algorithm we propose is of complexity $O(n \log n)$ and has an approximation ratio of 3/4 under some condition. In addition to developing the FCF algorithm, we implement an NFV platform equipped with two types of cores. We validate the effectiveness of our method, through implementation and measurement. Besides, extensive simulation results show that our algorithm performs very well in load-balancing factor, the number of used cores, and the sum of excess usage.

Acknowledgment

Jung-Chun Kao is the corresponding author. This work was supported in part by National Science and Technology Council, Taiwan, under grants no. NSTC 112-2221-E-007-075.

References

[1] J. Csirik, J. B. Frenk, M. Labbe, and S. Zhang, "Two simple algorithms for bin covering," Acta Cybernetica, vol. 14, no. 1, Feb. 1999.

[2] C.-F. Kuo, "NFV performance prediction based on run-time instruction analysis," master thesis, National Tsing Hua University, July 2021.

[3] W. Zhang, G. Liu, W. Zhang, et al., "OpenNetVM: A Platform for High Performance Network Service Chains," in Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, 2016.

[4] K. Wiles, The Pktgen Application, https://pktgen-dpdk.readthedocs.io/en/latest.

[5] C.-Y. Lee, "Resource allocation for NFV in hybrid multi-core architecture," master thesis, National Tsing Hua University, Jul. 2022.

[6] Catherinemeadows, The Simple Forward, https://github.com/sdnfv/openNetVM/tree/master/examples/simple_forward.

[7] Catherinemeadows, The AES encryption, https://github.com/sdnfv/openNetVM/tree/master/examples/aes_encrypt.

[8] J. Zheng, C. Tian, H. Dai, et al., "Optimizing NFV chain deployment in software-defined cellular core," IEEE Journal on Selected Areas in Communications, vol. 38, no. 2, Feb. 2020.

[9] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "A green VNFs placement and chaining algorithm," IEEE/IFIP Network Operations and Management Symposium, 2018.

[10] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," International Conference on Network and Service Management (CNSM), 2015.

[11] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "Energy efficient algorithm for VNF placement and chaining," IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2017.

[12] I. Jang, S. Choo, M. Kim, S. Pack, and M. K. Shin, "Optimal network resource utilization in service function chaining," IEEE NetSoft Conference and Workshops (NetSoft), 2016.

[13] X. Zhao, X. Jia, and Y. Hua, "An efficient VNF deployment algorithm for SFC scaling-out based on the proposed scaling management mechanism," Information Communication Technologies Conference (ICTC), 2020.

[14] S. Qi, S. Li, S. Lin, M. Y. Saidi, and K. Chen, "Energy-efficient VNF deployment for graph-structured SFC based on graph neural network and constrained deep reinforcement learning," Asia-Pacific Network Operations and Management Symposium, 2021.

[15] R. D. Thomas Willhalm, Intel Performance Counter Monitor, https://github.com/opcm/pcm.

[16] Dhiru Kholia, Dynamic Instrumentation Tool Platform (DynamoRIO), https://github.com/DynamoRIO.